CS112 - Fall 2022
Lab03
Instructor: Paul Haskell

READING

In the Java Software Solutions textbook…

- The reading for this lecture is sections: 2.4-2.5, 5.1-5.4
- The reading for the next lecture is sections: 3.1, 4.1-4.4 (but ignore UML)

INTRODUCTION

In this lab you will write some programs that use Java's built-in numerical data types. You will explore the limits of the data types. Although people who use computers to do mathematical computations often expect perfect results, there are limitations in both the range and accuracy of computer mathematics. A few times when programmers forgot these limitations, the result was a spectacularly famous failure.

Explosion of the Ariane 5

On June 4, 1996 an unmanned Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after lift-off. The rocket was on its first voyage, after a decade of development costing \$7 billion. The destroyed rocket and its cargo were valued at \$500 million. A board of inquiry investigated the causes of the explosion and in two weeks issued a report. It turned out that the cause of the failure was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer. The number was larger than … the largest integer storeable in a 16 bit signed integer, and thus the conversion failed.[1]

Limits of Fixed Point Data Types

Java's built-in data types cannot represent arbitrary large (positive or negative) numbers, and the "smaller" the data type, the smaller the limits. The first assignment in this lab is to calculate the limits for byte, short, and int data types. One way to find these limits is to start with a `long` variable `var1`. Convert the `long` to a shorter type e.g. `byte`. Then convert it back to a new `long` variable `var2`. Compare `var1` to `var2`—if they are not equal, then the shorter type cannot handle the value of `var1`.

Now we start with `var1` equal to 0 and increase it repeatedly by 1 until we have a problem. Use a while() loop to do the checking and incrementing.

How can you find the limits in the negative direction?

[1] https://www-users.cse.umn.edu/~arnold/455.f96/disasters.html. Downloaded 29 June 2022.

For this lab you shall write a program called **Limits.java** that finds and prints the largest positive and "most negative" values that are properly supported by the `byte`, `short`, and `int` data types. (Don't report one past the last supported values.)

The output from your Limits.java shall be as follows:

```
Maximum byte value is <<value>>
Minimum byte value is <<value>>
Maximum short value is <<value>>
Minimum short value is <<value>>
Maximum int value is <<value>>
Minimum int value is <<value>>
```

Limits of Floating Point Data Types

There are entire graduate courses devoted to the study of the limits and accuracy of mathematical computations done with floating point data types. For this lab, we will take on the first homework problem from the course I took on this subject: we will find (approximately) the smallest positive nonzero number that can be represented with the `float` and `double` data types.

To do this, initialize a `float` (and later a `double`) variable to some positive value, say 1.0. Keep on dividing the number in half and comparing the result against zero. Once the computer cannot tell that the value of your variable is greater than zero, then you have your answer (the previous value, which can be distinguished from zero).

For this lab you shall write a program called **FloatLimits.java** that finds the smallest positive nonzero `float` and `double` values and prints them out. The output from **FloatLimits.java** shall be as follows:

```
Smallest positive float is <<answer>>
Smallest positive double is <<answer>>
```

Taylor Series

You might have learned in Calculus that any differentiable function can be approximated with a polynomial. The polynomial is called the Taylor Series for the function. The more accuracy you need in your approximation, the more terms you must include in your polynomial.

For small enough values of x, ln(1 + x) , the natural logarithm of (1+x), can be approximated by the following polynomial:

$$x - x^2/2 + x^3/3 - x^4/4 + \ldots$$

For this lab you shall write a program called **Taylor.java** that will calculate ln(2) a few different ways:

- First, use the `Math.log()` function to calculate the answer directly

- Next, use the Taylor Series. What value do you choose for x to calculate ln(2) with the above formula? What approximation value do you get with 10 terms of the Taylor series? With 100 terms? 1000?
- Use a `while()` loop and a "counter variable" (`int counter`) to control how many terms you use in your approximation.

- Your program must produce the following four lines of output (`System.out.println()` will be useful):
  ln(2) = <<correct answer>>
  Taylor(10) = <<sum of first 10 terms in Taylor series>>
  Taylor(100) = <<sum of first 100 terms in Taylor series>>
  Taylor(1000) = <<sum of first 1000 terms in Taylor series>>

Data Conversion

Here you will demonstrate some of the changes that happen when we convert "larger" data types to "smaller" ones. You shall write a program called **DataConvert.java** that will do such conversions and output the results.

First, initialize a float variable to 2.5. Then create an int variable and initialize the int with the float, via casting. Print "2.5 cast to int gives <<value of the cast>>"

Set the float to -4.5 and set the int to this value with another cast.
Print "-4.5 cast to int gives <<value of the cast>>"

Now create a double variable and initialize it to (1.0/3.0). Assign the float to the value of the double via a cast. Print "double 1/3 = <<value of double>> but float 1/3 = <<value of float>>"

Now assign the double variable the value 8.0e9. Create a long variable and initialize it with the value of the double via a cast. Next create an int variable and initialize it to the value of the double. Print "long value of 8.0e9 = <<long value>> but int value of 8.0e9 = <<int value>>"

Assign your int variable the value 256. Create a byte variable and initialize it with the value of the int. Print "byte value of 256 is <<value you got>>"

Now set the integer to 257, set the byte to the int value (via a cast, of course), and print "byte value of 257 is <<value you got>>"

Next set the integer to 258, set the byte to the int value (via a cast, of course), and print "byte value of 258 is <<value you got>>".

Finally set the integer to 511, set the byte to the int value, and print "byte value of 511 is <<value you got>>".

Not for credit, add a comment to your code after this println() statement with your thoughts on why you are seeing the pattern you see for the four byte values.

Reminder

Put useful comments into your code.

Put your programs into a subdirectory called **Lab03** inside your **MyWork** directory, and remember to push your **Lab03** to GitHub before the deadline. This assignment must be turned in before Monday Sept 5th at 11:59pm (even though it is not a school day).

Conclusion

In this lab you developed some Java programs on your own, from scratch. You saw some of the limitations of Java's built-in numerical data types. And you wrote a program that performed useful mathematics, to compute the natural logarithm of a number with and without using the built-in Math library.

Rubric

Each program is worth 10 points

- 2 points for a program in the proper directory, with the proper name, that compiles and runs.
- 6 points for correct output.
- 2 points for "good coding style". For this lab, good coding style means:
    - Some useful comments at the top of each file describing what the program does
    - Some useful comments by key statements in each program describing what the code does
    - Useful variable names
    - Reasonably organized layout of the software